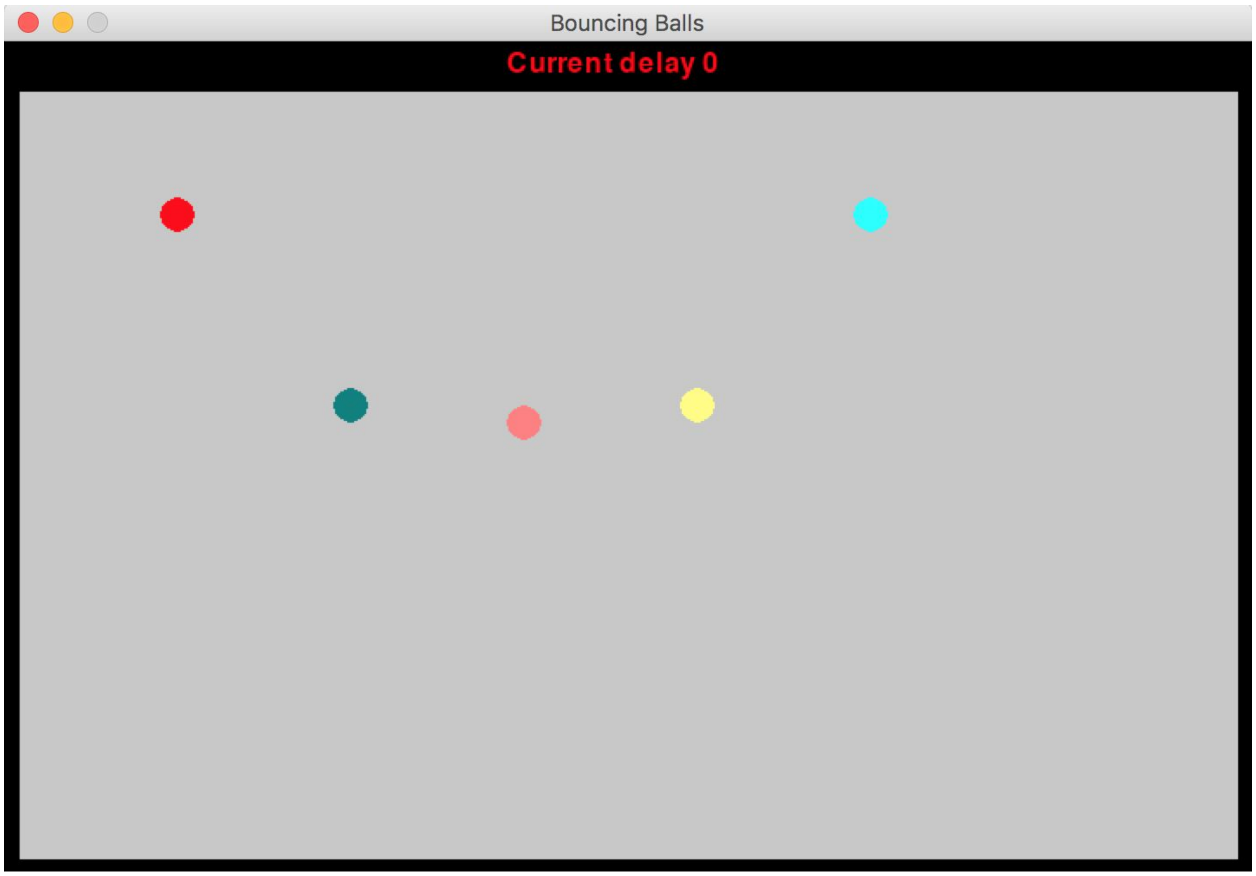


Project Case Study: Bouncing Balls

Year level band: 9-10

Description:

This project creates a program where five balls bounce at different rates, to build skill in the use of Object-Oriented programming design and implementation, and data representation. Students will learn about Object-Oriented programming using existing Classes and Objects, as well in designing and creating multiple new Classes, making this a good intermediate project.

Resources:

- Mac or PC with Python 3 and pygame library installed

Prior Student Learning:

An understanding of general and Object-Oriented programming concepts.

Digital Technologies Summary	This activity introduces students to: ● Object-Oriented design and implementation, including the analysis, tracing and understanding and existing Object-Oriented software components and the design and creation of new Object-Oriented software components. ● The lesson introduces the students to the idea of designing interactive programs, developing systems thinking, focusing on how different aspects of a program interact with each other.
Band	**Content Descriptors**
9-10	● Investigate the role of hardware and software in managing, controlling and securing the movement of and access to data in networked digital systems (ACTDIK034) ● Analyse and visualise data to create information and address complex problems, and model processes, entities and their relationships using structured data (ACTDIP037) ● Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics (ACTDIP039) ● Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040) ● Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041) ● Plan and manage projects using an iterative and collaborative approach, identifying risks and considering safety and sustainability (ACTDIP044)
Achievement Standards	● Students plan and manage digital projects using an iterative approach. ● They define and decompose complex problems in terms of functional and non-functional requirements. ● Students design and evaluate user experiences and algorithms. ● They design and implement modular programs, including an object-oriented program, using algorithms and data structures involving modular functions that reflect the relationships of real-world data and data entities. ● They share and collaborate online, establishing protocols for the use, transmission and maintenance of data and projects.

Project Outline

In this project, students will work through the problem solving and project design process to create the Bouncing Balls program, which simulates the effect of balls falling under the influence of gravity and then bouncing from the floor.

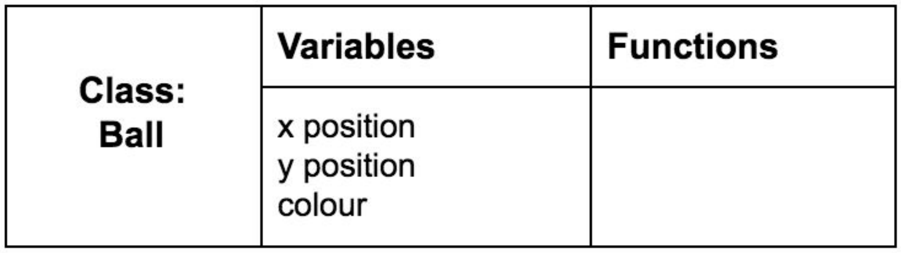
Defining the Problem	To create the **Bouncing Ball program** that draws five differently coloured balls on the screen, which will bounce as if they are under the influence of gravitational force. To produce an interesting pattern, the rate of fall is set differently across the balls. The balls follow the equations of motion and accelerate the longer they fall and decelerate as they climb back up. The user may use the arrow keys to increase or decrease the speed at which the animation runs, to allow them to inspect the movement of the balls. In this problem, we want to write a game where a graphical representation of five balls bounce on the screen. The balls start in a position as if they are up in the air. They then fall towards the bottom of the screen, bouncing back up when they strike the bottom, slowing down as they approach their old position and then repeating this activity. To write this program we are going to need: ● A way of representing the balls ● A way of representing the way that they fall ● A way to control the speed of animation ● An interface to display and control all of this Our system is going to involve working with software, so let's move on to starting the design of our software system. We are going to use an object-oriented approach and provide some detail here. The 9/10 MOOC course contains a much more detailed exploration of how to build this. There is additional context for this as we wish the balls to fall as if they are under the influence of a gravitational force. Things speed up as they fall under gravity, that is they accelerate as they fall. On Earth, after 1 second of fall, an object will be falling at 4.9m/s. After 2 seconds, it would be falling at 19.6m/s. From Physics, the equations of motion tell us that the distance travelled by any object under acceleration, which starts at rest, is given by: Distance travelled = (0.5 * acceleration * (time*time)) That is, the relationship increases by the square of time. On the way down, our ball will be accelerating towards the bottom and it will travel further and further every second. Once it bounces, the same gravitational force will slow it down in the same manner. In this software, we are going to simulate a bouncing ball, which should

appear to speed up and slow down as one would in the real world. To make the program visually appealing, we'll use a range of accelerations to produce patterns.

We have to think about the Classes that we want to build, with the associated variables and functions that will make sense for the development.

Let's start by looking at the Ball class. The first thing we can see is that it has to have its own position stored in a variable because we're going to create more than one Ball object - every object needs its own copy of the x and y position of the ball. We've talked about the colour of the ball as well so we probably need that.

That gives us an initial Class that looks like this.



Class: Ball	**Variables**	**Functions**
	x position y position colour	

But which functions do we want in there?

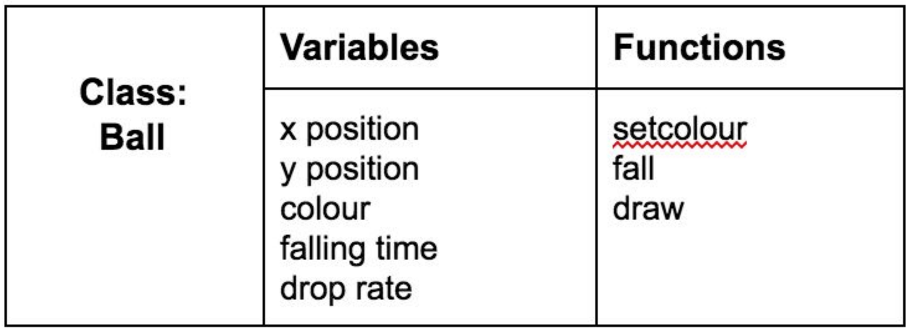
Here are the requirements (functional requirements) for how the ball is going to used.

1. The ball starts in a position on the screen
2. The ball is visible
3. The ball can have its colour changed
4. The ball will accelerate towards the bottom of the screen
5. The ball will bounce off the bottom and decelerate on the way up

Some thought on these behaviours will give us the likely functions: __init__ to set things up, setcolour to change the colour, draw so that we can see it, and fall to make it fall.

But we know two things: first that the speed at which the ball moves depends on how long it has been accelerating or decelerating, second that we have more than one ball and we wish to have some variation in the falling rate.

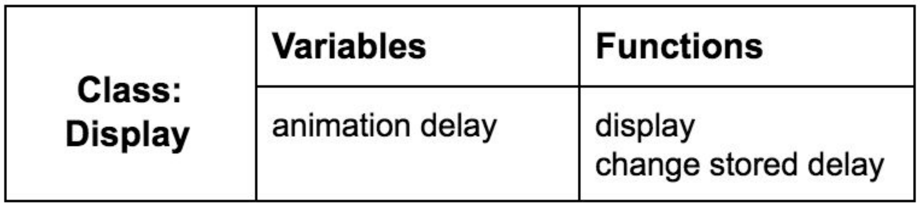
We probably need a variable in the Class to store how long the ball has been falling and another variable to keep track of the "local" gravity that we wish the ball to experience.



	Variables	Functions
Class: Ball	x position y position colour falling time drop rate	setcolour fall draw

We'll see later that we need one more variable, because of the way that we choose to implement it, but we'll come back to this. We certainly have enough to produce a few coloured balls and start them dropping under whichever gravity we give them.

We did say we wanted to control the speed of the animation and, whenever we offer a user control, we often want to offer them a way of seeing what's going on. Therefore, we need some way of displaying what the current animation delay is. We'll do this in a similar way to how we built the scoreboard in the worked example. In this case, we'll create a Class that will provide a way of storing the current animation rate and displaying it. That would look like this:



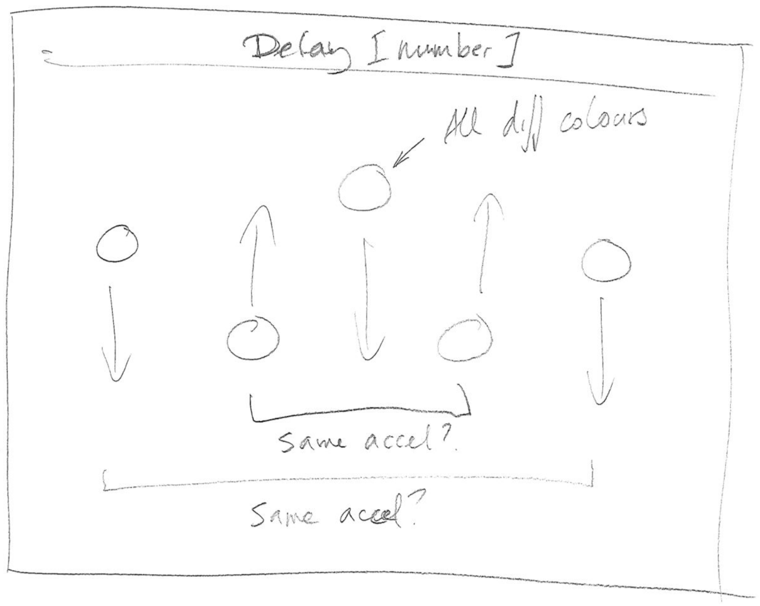
	Variables	Functions
Class: Display	animation delay	display change stored delay

Let's look at how a program to run the whole game might look:

1. Draw the playing area with bounding rectangle, set the delay to zero and display it.
2. Create some balls.
3. Draw the balls in their starting positions.
4. Start the balls falling.
5. If a user hits up arrow, increase the delay, but if a user hits down arrow, decrease the delay.

The balls aren't, of course, actually bouncing. We are using their position and the relative position of the walls to decide when to reverse direction. In a game, we refer to this detection of proximity as collision detection. Although the objects didn't really collide, in a real world sense, they are 'touching' and we can perform actions based on this.

We will return to a broader discussion of how we could extend this towards the end of this exemplar. For now, we will move onto working on the data of the program.

Understanding our Data	Most of the data representation in the program is straightforward. The animation delay is a number. The position of each ball is given by an (x,y) coordinate pair and its colour is also straightforward. Each ball will keep track of how it has been either accelerating or decelerating. Our Display class needs to be told when the delay has changed and this function will be called whenever the user hits a key. The whole program is a little more complex because we have multiple balls that we wish to animate. We could create a number of Ball objects directly, labelled ball1, ball2, ball3, etc, but this would require us to write a lot of duplicate code to animate the balls. Given that we wish to call the draw and fall functions in each Ball, we are going to create new Ball objects and put them into a Python list (similar to an array in other languages). That way, we can write small helper functions that will take a list of Ball objects and call the same function in all of them.
Prototyping and User Interface Design	Sketch prototypes are very useful to get a feel for how the game will work. These can be actual sketches or small examples using a whiteboard or post-it notes.  Delay [number] All diff colours Same accel? Same accel? This early sketch shows the rough layout on the screen, as well as the idea that we match some of the accelerations to produce a pattern as the balls bounce. You can see, from the arrows, that the balls could be travelling in different directions at different times. A prototype of the code of this game could be a single ball bouncing, or even a single ball falling as a very early example, before collision detection is active.

Implementation

The full Python3 code for this is shown below.

In this implementation, we have kept track of the direction that each ball is moving, up or down, in order to execute the correct change to y. This requires us to add another variable to the Ball class. You could also choose to make the ‘ftime’ variable positive or negative to simplify the logic but the ‘direction’ approach makes it easier for students to work on a falling ball first and then extend their logic to the other direction.

The following code shows the entire implementation. This is one way to solve the problem but there are many others.

```
# We need these libraries to access the pygame functions

import sys, pygame

pygame.init();     # Set up pygame
# Set up the font we need for the display
font = pygame.font.SysFont("None", 24)
# Construct the window
main = pygame.display.set_mode((720, 480))
# Fill it with grey
main.fill((200, 200, 200))

# This is the main class of this program: Ball

class Ball:
    x=0                # X coordinate
    y=0                # Y coordinate
    colr=(255,0,0)    # Colour
    direction=1        # 1: Ball going down,
                      #-1: Ball going up
    ftime=0           # Time in this direction
    droprate=0         # "local gravity"

    # Initialise the Ball with x,y,droprate.
    # Set falling time to 0
    def __init__(self,x,y,droprate):
        self.x=x
        self.y=y
        self.droprate=droprate
        self.ftime=0

    # Provide  way to change the colour
    def setcolour(self,newcolr):
        self.colr=newcolr

    # Draw a circle to represent the ball on the screen
    def draw(self):

pygame.draw.circle(main,self.colr,(self.x,self.y),10,0)

    # This handles all of the falling
    def fall(self):

      # First erase the old ball by painting over it with
      # the background colour


pygame.draw.circle(main,(200,200,200),(self.x,self.y),10,0)

      # If we're falling, then

        if self.direction==1:

  # Check to see if we're far enough away to keep falling
            if (self.y<=460):

  # Move y coordinate by the amount given by the formula
  # we mentioned earlier

self.y=int(self.y+(self.droprate*(self.ftime*self.ftime)))
                # increase the time spent falling by 1
                self.ftime+=1
            else:
            # We bounced! Change direction!

                self.direction=-1
                self.ftime-=1
        else:
      # We're bouncing back up

          # If we're not about to  hit the top
            if (self.y>=30):

              # Move based on our current speed


self.y=int(self.y-(self.droprate*(self.ftime*self.ftime)))
                self.ftime-=1
            else:
             # Start falling again
            #(this is a bounce off the top)
                self.direction=-1

        # If we use all of our old acceleration,
        # start falling again
        # and set our falling time to 1.

        if (self.ftime==0):
            self.ftime=1
            self.direction=1

# Helper function to call draw
# on every ball in a list.
```

```
def drawBalls(alist):
    for ball in alist:
        ball.draw()

# Helper function to call fall
# on every ball in a list

def fallBalls(alist):
    for ball in alist:
        ball.fall()

# The Display class creates objects
# that will display the current animation
# delay to users.

class Display:
    slowspeed = 0  #Animation delay

    def __init__(self):
        self.slowspeed=0 # Initially set to zero.

    # Allow it to be set from outside.

    def setslow(self,setspeed):
        self.slowspeed=setspeed

    # Show the animation delay at the top of the screen.
    def display(self):
        # Redraw the underlying rectangle to remove blur
        pygame.draw.rect(main, (0,0,0), ((0,0),(720,20)),0)
        # Produce a text object to display
        text=font.render("Current delay "+
str(self.slowspeed), True,(255,0,0))
        # And display it
        main.blit(text,(290,5))

# The Main() routine sets everything up

def Main():

    # Set animation delay to 0
    slowdown=0

    # Create and setup the Display
    display = Display()
    display.setslow(slowdown)
    display.display()

    # Create all of the balls
    # First create the list to hold them
    balllist=[]

    # Then append Balls in different places to the list
    # The third number is the "local gravity" setting
    # Which means that the ball in the centre is
    # roughly at 1 Earth Gravity, but 2 and 4 are
    # at half that, and 1 and 5 are at half that again

    balllist.append(Ball(100,100,1))
    balllist.append(Ball(200,100,2))
    balllist.append(Ball(300,100,4))
    balllist.append(Ball(400,100,2))
    balllist.append(Ball(500,100,1))

    # Balls are red by default.
    # Using list indexes, we recolour 4 balls.

    balllist[1].setcolour((0,128,128))
    balllist[2].setcolour((255,128,128))
    balllist[3].setcolour((255,255,128))
    balllist[4].setcolour((0,255,255))

    # We call the helper function to display them

    drawBalls(balllist)

    # And force pygame to refresh the screen
    pygame.display.update()

    # Now we go into the game loop
    while (True):
      # Get all of the balls to fall
        fallBalls(balllist)

        # Redraw them
        drawBalls(balllist)

    # There is some arithmetic rounding that can make the
    # screen a little messy. Redraw the boundary to
    # give the illusion of a seamless bounce.

        pygame.draw.rect(main, (0,0,0), ((0,20),(720,460)),
16)
      # Show the display
        display.display()
      # Force the screen to update
        pygame.display.update()

      # Now see if the user has done anything
        for event in pygame.event.get():

          # If the user hits a key
            if event.type == pygame.KEYDOWN:

              # If it's up arrow, increase delay
                if event.key == pygame.K_UP:
                    slowdown+=10
              # If it's down arrow, decrease delay
                elif event.key == pygame.K_DOWN:
                # But not below zero!
                    if (slowdown>0):
                        slowdown-=10
                # Now update the display for the
            # new value of delay
                display.setslow(slowdown)

          # If the user closes the window, quit
            if event.type == pygame.QUIT:
                pygame.quit(); sys.exit()

      # And update the animation delay where
      # actually changes the way that the
      # Python code animates
        pygame.time.delay(slowdown)

# When the program first runs

if __name__ == '__main__':

    # Label the screen and draw the boundaries
    pygame.display.set_caption("Bouncing Balls")
    pygame.draw.rect(main, (0,0,0), ((0,0),(720,20)),0)
    pygame.draw.rect(main, (0,0,0), ((0,20),(720,460)), 16)

    # Then call the Main routine
    Main()
```

Testing and Evaluation

How can we test this code?

The functional requirements that the student developed turn, almost immediately, into a checklist that can be handed to another student. Let's look at an example.

The first thing we want to do is to make sure that we can draw at least one ball and see it fall on the screen. So our testing for correct function will be:

1. Can I display the ball?
2. Does it move towards the bottom of the screen?
3. Does it appear to speed up?
4. What happens when it hits the bottom?

If we identify an error in the Ball, because it's a Class, we will go into the Ball class and fix it there. The next step for the Ball will be checking what happens when the Ball collides with the bottom of the screen. Does it change direction? Does it slow down as it climbs? Does it reach the same height as before? Can we produce multiple balls at different fall rates with different colours? We'd then continue to test the program until we've tested all of the individual elements and their interactions together.

	One useful test case is to see if everything is being drawn where you expect. Are the balls are starting at the same point and then falling the correct way? Testing the non-functional requirements often falls into the realm of playability. We noted before that we can ask students whether the controls are obvious and responsive, and whether the animation delay changes slow the game down.
Project Extensions	There are many ways this could be extended. Some that we have thought of include: ● Could you make the balls move in the x direction as well? How do they bounce and could you make them collide with the sides? ● Can you make this more realistic by modelling a ball that isn't perfectly elastic - one that slows down? ● Can you introduce the balls at lots of different angles and have them collide with each other?

Project Reflection

In this project, students will undertake an extensive design and implementation process. The project is well suited to be taught over a number of sessions, and using group work. While we have not included explicit assessment information here, we have provided a discussion of how these sessions might be structured, and how students might be able to demonstrate their learning.

For a more extensive discussion of assessment possibilities within Digital Technologies at this level, please join us in our online community and course: www.csermoocs.adelaide.edu.au

Learning construction	Students work independently or in teams to design and implement the Python program. This project is well suited to teamwork, and can be used to build collaboration and communication skills. The activity is about experimenting, trying new solutions, and debugging. Once students have successfully built a working model of the bouncing balls, ask them to explore what other physics or real work influences they could add to the program to make it more realistic.. The project can be extended in multiple ways through exploring different variants of the game and customising the components. Encourage students to help each other - and look for help on the internet. Ask a friend. Ask Google. Then ask the teacher.
Learning demo	While students are working in groups, ask questions to give them the opportunity to demonstrate their thinking and understanding: What challenges have you faced in developing this design?

	How did they decide how to store the data? How do the objects interact with each other? What would happen if you bounced the ball at an angle?
Learning reflection	Remind students that this is a very simple program but that the program we have been developing has a development process that you would find in real-world game development. Simulation of the real world is one of the most useful aspects of computing and, although it is very simple, this is a simulation of how things bounce in the real world. The steps that they have taken can be applied to many other larger projects ● Can you think of any simulations that they want to build that they might use to build games, like Pool, Billiards or Snooker, for example? ● What would the addition of sound do to this project? ● Can we use this to predict things in the real world? What other real-world problems could they solve?

Teacher/Student Instructions:

Download the latest version of Python 3 from http://python.org .
When this has been installed, follow the instructions at http://pygame.org/wiki/GettingStarted to install the Pygame library.
IMPORTANT: You must use python3 to run the pygame library on many platforms.

CSER Professional Learning:

This Case Study supports learning from the CSER 9&10 Digital Technologies: Explore! MOOC. You can join us here to learn more about Digital Technologies in Years 9 & 10: www.csermoocs.adelaide.edu.au

Further Resources:

1. Information about Python, including tutorials: http://python.org
2. Information on the pygame library: http://pygame.org